

Neural Network in Python Summary

Kuo Lung Chien
簡國龍

February 27, 2021

2LayerNeuralNetwork.py

- 定義 input(X) 和 output(Y)

```
X = np.array([[0,0,0],[0,0,1],[0,1,0], \
              [0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]], dtype=float)\\
y = np.array([[1], [0], [0], [0], [0], \
              [0], [0], [1]], dtype=float)
```

- 設定神經元及權重

```
def __init__(self):
    # X1,X2,X3(自訂義 input 的神經元數量)
    self.inputLayerSize = 3
    # Y1(自訂義 output 的神經元數量)
    self.outputLayerSize = 1
    # Size of the hidden layer(自訂義 hiddenLayer 的神經元數量)
    self.hiddenLayerSize = 4

    # 設定第一層權重為隨機數值, input—>hidden
    self.W1 = \
    np.random.randn(self.inputLayerSize, self.hiddenLayerSize)

    # 設定第二層權重為隨機數值, hidden—>output
    self.W2 = \
    np.random.randn(self.hiddenLayerSize, self.outputLayerSize)
```

- feedForward(前饋)

```
def feedForward(self, X):
    """
    第一層的動態方程式(activation function)輸入(z)
    z(activation function)=
    第一層所有神經元的 input*weights 總和輸入到
    第二層的其中一個神經元
    """
    self.z = np.dot(X, self.W1)

    """
    第一層的動態方程式(activation function)輸出(a)
    z2(a) = 動態方程式(activation function)用 Sigmoid function 算法
    """
    self.z2 = self.activationSigmoid(self.z)

    """
```

```

第二層的動態方程式 (activation function) 輸入 ( $z$ )
 $z3(activation\ function) =$ 
第二層所有神經元的  $input * weights$  總和輸入到
輸出層的 (其中一個) 神經元
"""
self.z3 = np.dot(self.z2, self.W2)

"""
第二層的動態方程式 (activation function) 輸出 ( $a$ )
 $o(a) =$  動態方程式 (activation function) 用 Sigmoid function 算法
"""
o = self.activationSigmoid(self.z3)

# 回傳出前饋結果
return o

```

- backwardPropagate(反向傳播)

```

def backwardPropagate(self, X, y, o):
    # 計算輸出誤差
    self.o_error = y - o

    # 利用 Sigmoid function 微分一次來修正輸出誤差 (錯誤、error)
    self.o_delta = self.o_error * self.activationSigmoidPrime(o)

    # 隱藏層的輸出誤差 * 權重
    self.z2_error = self.o_delta.dot(self.W2.T)

    # 將 Sigmoid function 算法用在隱藏層輸出誤差 (錯誤、error)
    self.z2_delta =
    self.z2_error * self.activationSigmoidPrime(self.z2)

    # 修正第一層權重數值,  $input \rightarrow hidden$ 
    self.W1 += X.T.dot(self.z2_delta)
    # 修正第二層權重數值,  $hidden \rightarrow output$ 
    self.W2 += self.z2.T.dot(self.o_delta)

```

- trainNetwork(訓練流程)

```

def trainNetwork(self, X, y):
    # 前饋循環
    o = self.feedForward(X)
    # 反向傳播值
    self.backwardPropagate(X, y, o)

```

- activationSigmoid

```

def activationSigmoid(self, s):
    # activation function
    # 使用 Sigmoid function 算法 (S-curve)
    return 1/(1+np.exp(-s))

```

Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- activationSigmoidPrime(sigmoid function 一次微分)

```
def activationSigmoidPrime(self, s):
    return s * (1 - s)
```

First derivative of Sigmoid function

$$\sigma(x)' = \sigma(x)[1 - \sigma(x)]$$

- saveSumSquaredLossList(儲存損失函數值)

```
def saveSumSquaredLossList(self, i, error):
    lossFile.write(str(i) + ", " + str(error.tolist()) + '\n')
```

- saveWeights(儲存權重值)

```
def saveWeights(self):
    np.savetxt("weightsLayer1.txt", self.W1, fmt="%s") #第一層權重
    np.savetxt("weightsLayer2.txt", self.W2, fmt="%s") #第二層權重
```

- predictOutput(結果輸出)

```
def predictOutput(self):
    print("Predicted XOR output data based on trained weights:")
    print("Expected (X1-X3): \n" + str(xPredicted))
    print("Output (Y1): \n" + str(self.feedForward(xPredicted)))
```

- Epochs(疊代次數, feedForward+backproagation 運算完算一次疊代)

```
# 訓練疊代次數
trainingEpochs = 1000
```

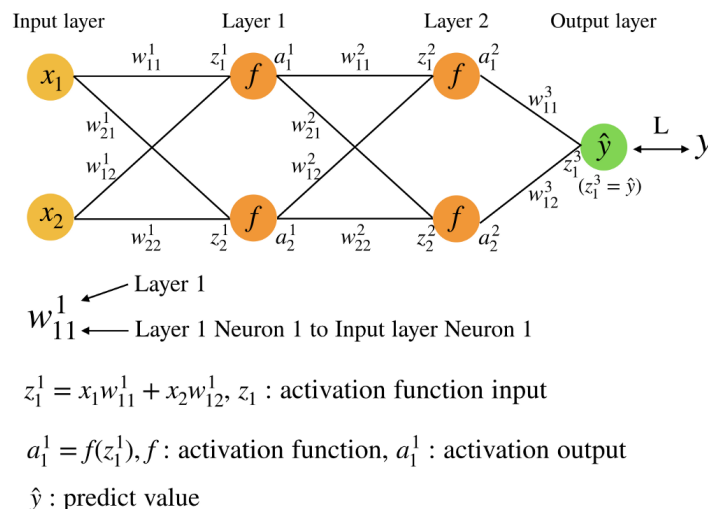


圖. 1: Neural network

TensorFlowKeras.py

- 定義 input(X) 和 output(Y)

```
X = np.array([[0,0,0],[0,0,1],[0,1,0],\
              [0,1,1],[1,0,0],[1,0,1],[1,1,0],[1,1,1]], dtype=float)
# X是三輸入XOR邏輯閘

y = np.array([[1], [0], [0], [0], [0],\
              [0], [0], [1]], dtype=float)
# Y是輸出神經網路
```

- 神經網絡設定

```
model = tf.keras.Sequential()
#sequential定義model為層狀結構

model.add(Dense(4, input_dim=3, activation='relu', use_bias=True))
'''
add是從最上層開始加入，Dense是密集連線的神經網路，
4:輸出空間(神經元，輸出到4個神經元)，input_dim:輸入神經元個數，
activation:定義啟動函數使用的類型，use_bias:使用偏差，True開啟。
從inputlayer輸出到hiddenlayer的設定
'''

#model.add(Dense(4, activation='relu', use_bias=True))
model.add(Dense(1, activation='sigmoid', use_bias=True))
# 從hiddenlayer輸出到outputlayer的設定

model.compile(loss='mean_squared_error', optimizer='adam',\
              metrics=['binary_accuracy'])
'''
配置訓練模組，loss function:用mean squared error(差平方誤差)，
optimizer:優化器，用adam function，
metrics: 計算準確率，用binary_accuracy
'''

print (model.get_weights())#印出回傳的正確權重

history = model.fit(X, y, epochs=2000, validation_data = (X, y))
'''
訓練模型給予固定epochs，迭代收集到的資料，
validation_data: 評估準確率(不包含在訓練裡面)
'''
```

ReLU

- *max_value*: 輸出後最大值上限
- *negative_slope*: 負斜率係數
- *threshold*: 可通過的數值界線

$$f(x) = \max(0, x)$$

Mean squared error(MSE)

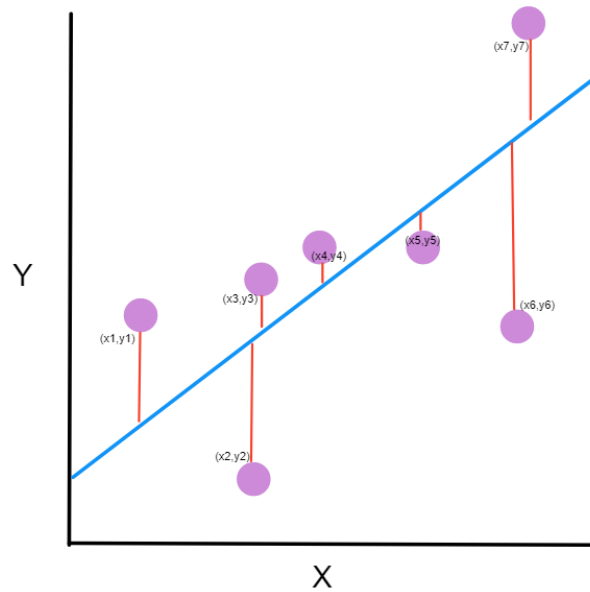


圖. 2: Mean Squared Error

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2$$

- 紀錄

```
model.summary()  
# 摘要資料使用在分析和可視化，為了確認訓練架構在符合預期方向  
  
loss_history = history.history["loss"]  
#回傳紀錄事件(loss)到history物件，取得fit方法的模組回傳值  
numpy_loss_history = np.array(loss_history)  
#將loss_history數值存成array  
np.savetxt("loss_history.txt", numpy_loss_history, delimiter="\n")  
#將numpy_loss_history存成loss_history.txt，並將每筆資料用換行符號隔開  
  
binary_accuracy_history = history.history["binary_accuracy"]  
#回傳紀錄事件(binary_accuracy)到history物件  
numpy_binary_accuracy = np.array(binary_accuracy_history)  
#將binary_accuracy_history數值存成array  
np.savetxt("binary_accuracy.txt", numpy_binary_accuracy, delimiter="\n")  
#將numpy_binary_accuracy存成binary_accuracy.txt，並將每筆資料用換行符號隔開
```

- 結果

```
print(np.mean(history.history["binary_accuracy"]))  
#印出平均binary_accuracy記錄到的數值  
result = model.predict(X).round()  
#替輸入樣本產生輸出預測  
print(result)  
#印出結果
```